Juan Cardenas

MM3 Final

MM3 Final:

Simulation of Solid and Liquid Argon

1. Necessary methodology and algorithm

I began this code, by trying to analyze the code that was sent to us. After a while, I decided to start almost from scratch. I began by setting up the environment to simulate 3D molecular dynamics. For this, I used ***visual python 7, the newest version*** of visual python. This is a python package that allows one to simulate physical events by giving them a virtual space in which to calculate and display objects under the influence of forces. Once I became proficient In setting up and manipulating objects in the virtual environment, I began to write loops to place particles in a Face centered cubic arrangement, in the simulation region. To do this, I created a function called, “**create\_solid()**” that would arrange the atoms in a face centered cubic lattice and append an array that contained all the particle positions, as well as their velocities, which I set to 0 for all particles. This function also added the atoms to the simulation display.

Once the lattice was assembled, I needed to add the dynamics to the system. To apply the forces that one particle exerts on all the others, I used the framework of the function used to calculate accelerations that was given to us in the code called “**nbody()**”, but instead of the parameters that were given to calculate the force, I inputted the new parameters that you had given us, new distance scale, mass, energy, etc. Once I created the modified “nbody()” function that returned an array of the acceleration of all particles, I needed to apply this acceleration to each particle to make them move in the simulation. To continue making the program more “modular” and update the positions of the particles, I created a function called “**new\_position\_velocity()”** that would take in the array of the current particle positions and velocities, and apply the acceleration to them, effectively updating their positions and velocities using the “**nbody()”** function. To update the position and velocities, I used ***the velocity-verlet algorithm.*** This is an algorithm that uses the forces due to the *current* particle position, to update the particle position, but uses the average of the forces between our current particle positions, and the forces between particles in the new positions incremented by our time step dt. The formula for the velocity-verlet algorithm are as follows.

Notice how the velocity is calculated after the position has been calculated because we need to use both the acceleration due to the particle arrangements at “t” as well as at “t+dt”. Using the velocity-verlet method to calculate my trajectories greatly smoothed out my particle movement and in general made the system more stable.

Within the **“new\_position\_velocity()”** function, I also calculated the mean square displacement of the particles by calculating the difference between the position r(t) and r(t+dt) of each particle and averaging it with respect to all the particles in my simulation box. The **“new\_position\_velocity()”** function then outputted an array for the mean square distance of the particles in the system for all increments of time from 0 to my final run time to be plotted.

From here I moved on to writing a program to produce the pair correlation function, called “***pair\_correlation\_function\_(liquid/solid)()”***. For this I used the formula in the slides you gave us.

Where rho is the total number of particles divided by the volume of our simulation box. For this I created a for loop to loop through all distances between .001 and LH which is half the length of our simulation box. And for each R, I checked if any particle separation distances fell between r+dr and r-dr if they did I would add a to a particle counter. The number stored in the variable named “count” would then be used to calculate the pair correlation function for a given R. This function returned two arrays that were the x and y values for the pair correlation function and the r value.

I was now ready to simulate the system, after a few failures (particles exploding violently into the void), I realized that I needed to set the appropriate box length as well as the correct particle distances so that particles were not too confined by the box, or were too close or far from each other. Having either would make the particles start in a state of high potential energy that would eventually turn into kinetic energy, leading the particles to move very quickly as though they were in a gas. Knowing that the equilibrium distance between any two particles in a face centered cubic arrangement should be 1.225, I set the distance between the corners of our face centered cubic lattice (*red dots in simulation*) to be a distance of 1.77 apart. This made it so the distance between any nearest neighbor to the particle centered on a face, was at a distance of 1.225 away (the equilibrium distance). Once the simple cubic spacing was set to 1.77 the particle reached a stable equilibrium and I could begin to adjust the box size, so that we could run a simulation which seemed to be part of an infinitely bigger solid. I determined the approptiate box size for this to be, If the lattice was at the center of the box to be ﻿“***dimension\*lat\_c + .5\*lat\_c”***  where dimension is the number of particles along one edge of a simple cubic lattice.

For the liquid simulation, I created a function called ***“create\_liquid()”*** this function spread the particles in our N by N lattice of the solid randomly around a later box (necessary for stability) and gave then starting velocities even spread from -1 to 1 in a gaussian distribution.

To start the simulation, I put the update function in a while loop as such.

﻿**print("=== Now starting simulation ===")**

r, v, atoms = create\_solid(r, atoms, dimension, lat\_c, radius, N)

while(t<runtime):

#vp.rate(2000)

r[r > ((dimension-1)\*lat\_c)/2 + HL] -= L # periodic bc

r[r < ((dimension-1)\*lat\_c)/2 - HL] += L

a = nbody(r, N)

r, v, atoms, mean\_sqr\_array = new\_position\_velocity(r, v, a, atoms,mean\_sqr\_array, N,dt)

t+=dt

**print("=== End of simulation ===\n\n")**

Esentially, what this code does is, create a lattice or liquid, check if any of the particle positions are out side of the bounds of the box, if they are, then add or subtract L to bring the particle to the other side of the box, simulating an infinite box. We then compute the acceleration due to the force felt by the particles due to their arrangement, and then update their position and velocity according to the velocity-verlet method. From there, we increment our time by dt and keep running until we have reached the time we specified by the variable “runtime”.

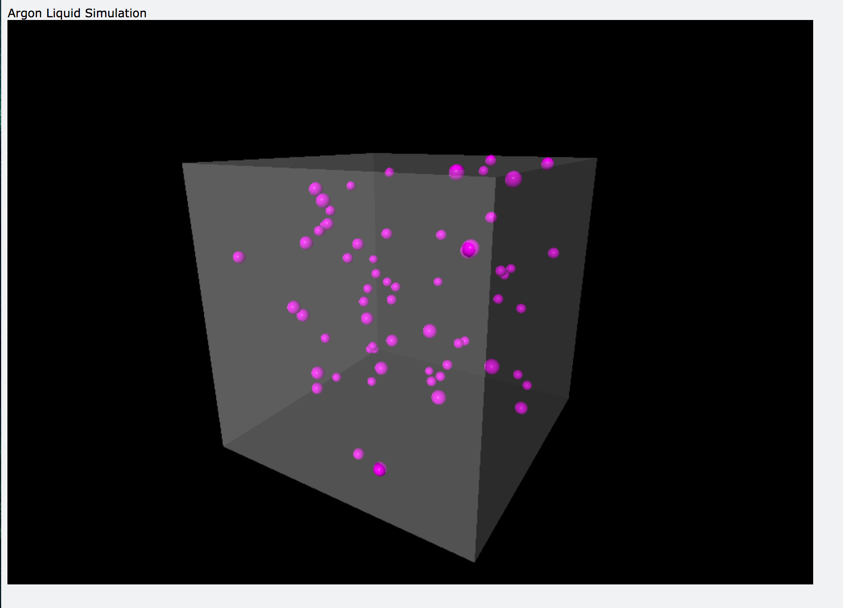
The same methodology goes for the liquid simulation, but instead we call the create\_liquid() function.

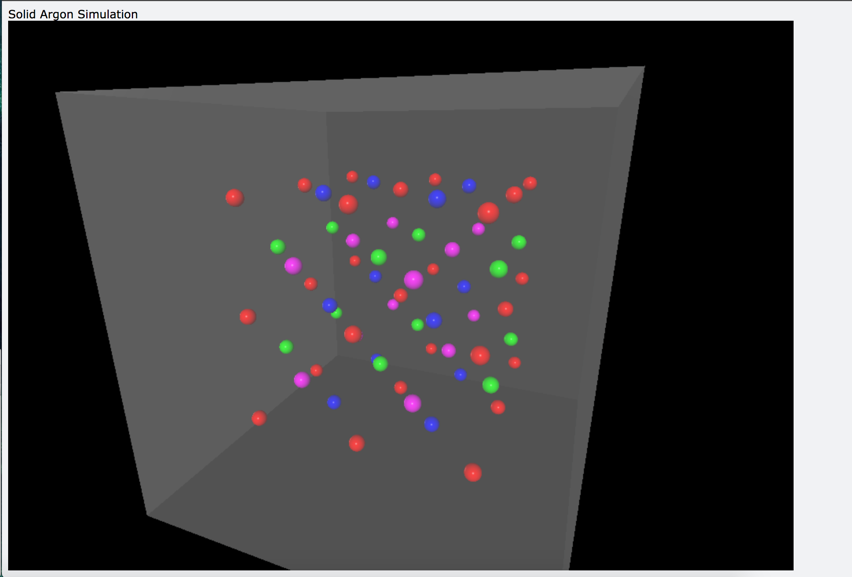
1. The Python code

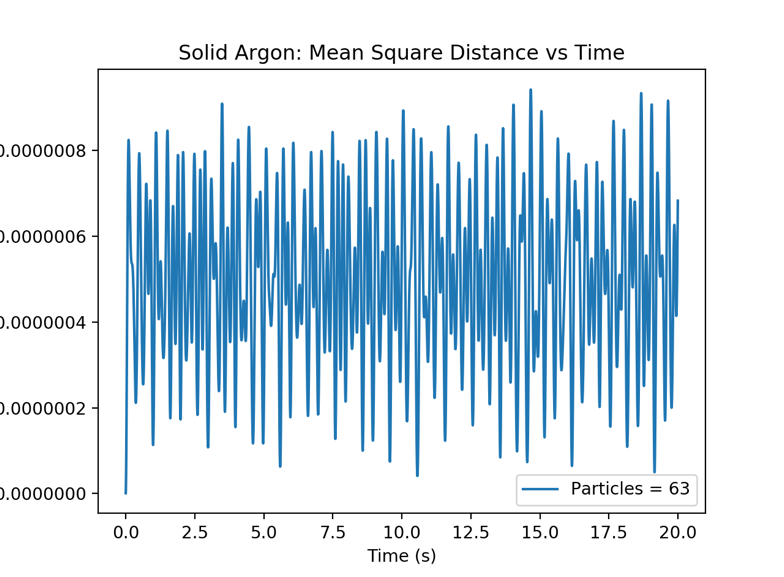
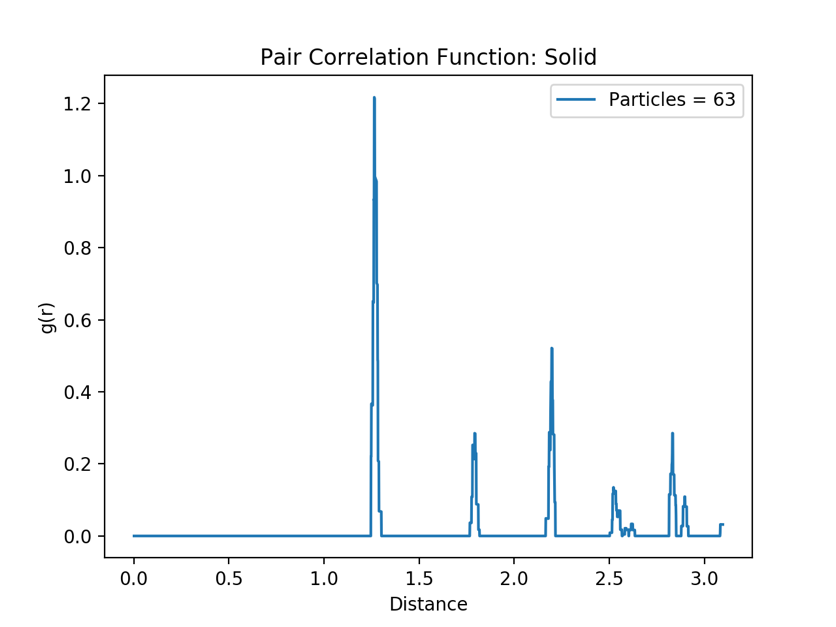
The python code is attached

### IMPORTANT### The code is NOW written in ***Python 3***

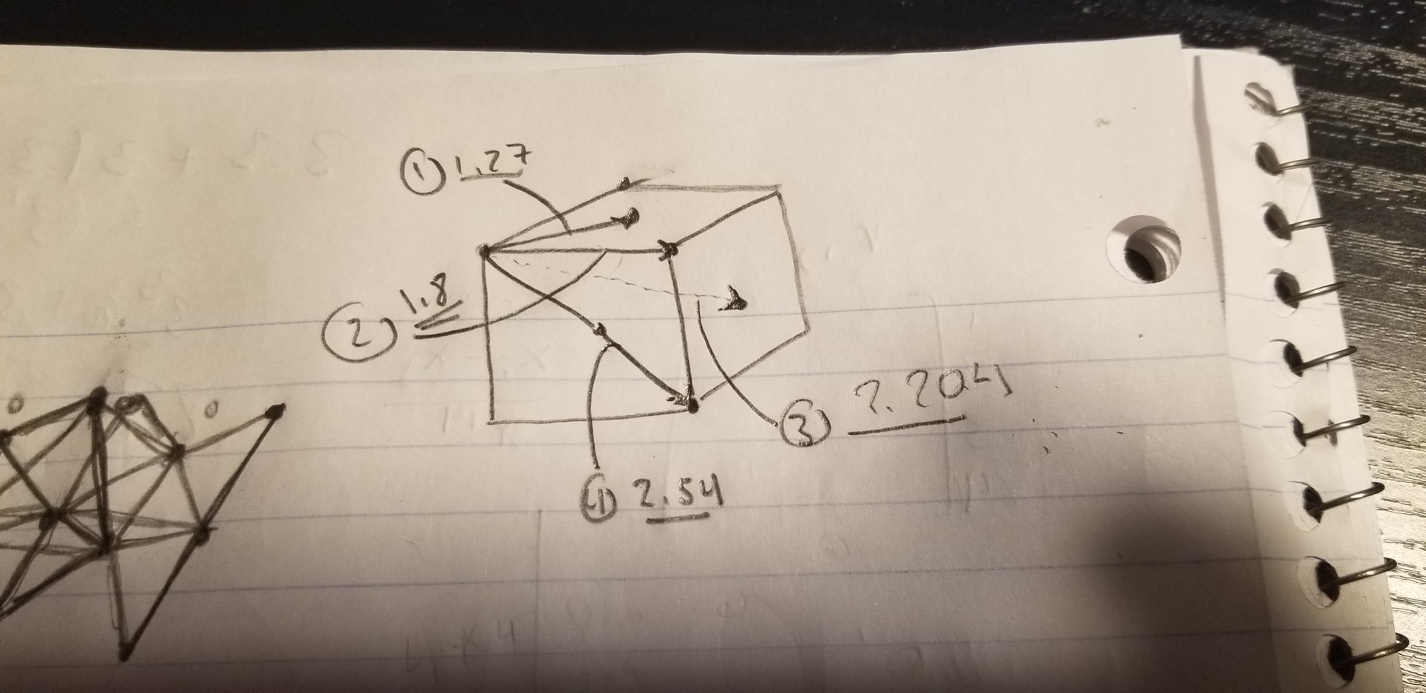
***\*Results and plots below\****

1. Results, plots and analysis



Here are a few pictures of the simulation. A 3x3 lattice is all I could manage in a reasonable computing time. As particles were added, the compute time would drastically increase. Below are the mean square distance plot with respect to time and the pair correlation function of the solid with respect to distance.

**SOLID:** We can see that the mean square distance for the solid averages at about .00000006, or about 0. This means that the particle are not beginning to accelerate and are therefore in a stable configuration. As for the pair correlation, this is exactly what we would expect from a solid, there are discrete points on the plot that indicate a large number of particles at a given distance from any given particle, this is characteristic of a solid because it is very ordered. In solids there are gaps where particles are not found followed by periodic distances where particles are found. The points on the plot correspond to the positions of a nearest neighbor when the atoms are arranged in a FCC lattice. As in the drawing below

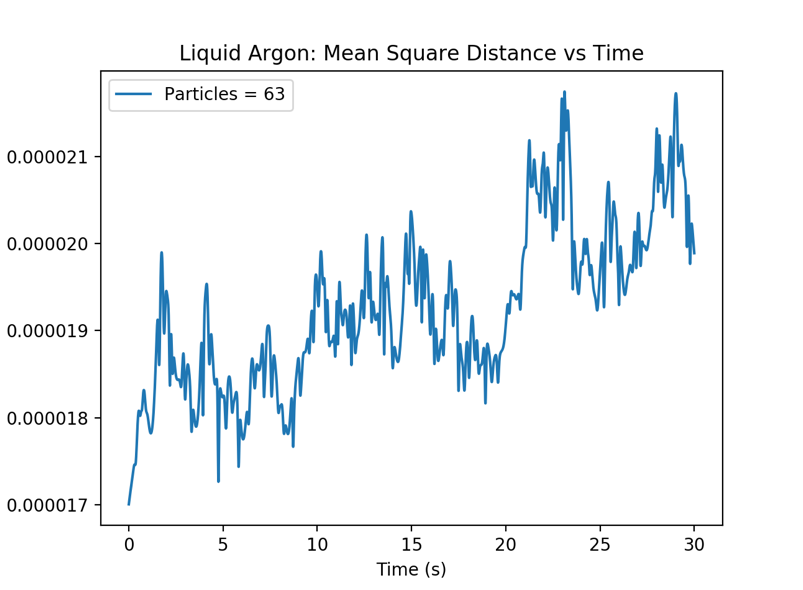
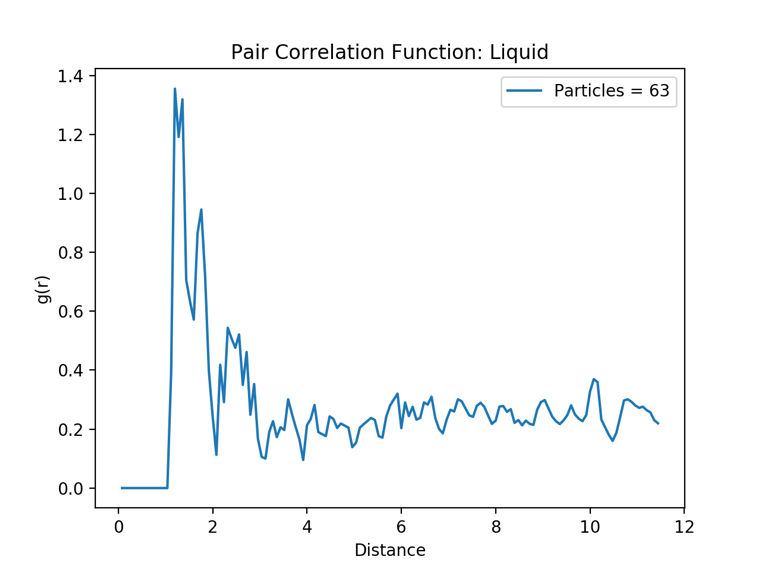


For this particular simulation I set the distance between cube corners to be 1.8, this results in the distances of particles in the diagram to correspond to the peaks in the graph.

**LIQUID:**

As for the liquid, this gave more variable results, and was highly dependent on the size of its container. Small changes in the size of the container would result in particles moving at very high velocity influencing all the others, eventually turning the liquid into a gas. So it was critical to find a “correct” box size. I would also encounter times where I would use the same initial conditions, and get completely different plots. As well as simulations, at times the liquid was calm and flowing, at other times a single particle would move rapidly and perturb the system enough to cascade it into a gas. **I suspect that the problem arises in the way that the particle positions are being placed randomly.** Since I am using a function to place the particles at any position from .01 to L-.01 some **particles may be randomly placed very close to each other**, forcing them to repel each other with great force once the simulation begins.

If this is the case, I ask you, please, Run the simulation once more to obtain plots that are similar to the ones I am going to display. They WILL appear.

Below are the Mean square distance and pair correlation function plots for a more common run of liquid argon.

We can see in the mean square distance plot that there is a trend towards linear growth which is representative of a particle undergoing a phase transition from a solid to a liquid. A change in a particle’s displacement over time indicates that there is on average, there is acceleration amongst the particles.

For the **pair correlation function**, the plot is characteristic of a liquid, there is a large amount of particles near 1.5 and then there is an oscillatory trend toward an equilibrium value of .2. Plots like these with an initial sharp peak and subsequent smaller peaks are indicative that the substance is in a liquid like state.

1. Conclusions

In conclusion, from the given simulation data we can see that the Leonard-Jones potential leads to a FCC lattice arrangement for low temperature (low energy) systems, and can accurately produce a liquid when particles are placed randomly with small velocities in a container. The plots that I have created from the data show a clear relation to solids and liquids that we see in real life. Although, there are still some problems that need to be solved, such as finding the correct cube size to particle number ratio in order to produce a reliable liquid state.